Name of Student: Pushkar Sane			
Roll Number: 45		Lab Assignment Number: 3	
Title of Lab Assignment: To implement Python programs using List, String, Set and Dictionary.			
DOP:		DOS:	
CO Mapped: CO1	PO Mapped: PO3, PO5, PSO1, PSO2	Signature:	Marks:

Practical No. 3

Aim: To implement Python programs using List, String, Set and Dictionary.

Description:

In Python, `List`, `String`, `Set`, and `Dictionary` are all built-in data types that serve different purposes and have unique characteristics. Here's a description of each with examples:

1. **List**
   - A list is an ordered collection of items, and it can contain elements of different data types.
   - Lists are mutable, meaning you can change their contents after they are created.
   - Lists are defined using square brackets `[ ]` and can contain zero or more elements separated by commas.
   - Example:
     my_list = [1, 2, 3, 4, 5]
     fruits = ["apple", "banana", "cherry"]
     mixed_list = [1, "hello", True, 3.14]

   Some Basic List operations

   a. **Accessing Elements**
      Example:
      first_element = my_list[0] # Access the first element (1)
      last_element = my_list[-1] # Access the last element (5)

   b. **Slicing**
      Example:
      sliced_list = my_list[1:4] # Creates a new list (2, 3, 4)

   c. **Appending and Extending**
      Example:
      my_list.append(6) # Adds 6 to the end of the list
      my_list.extend([7, 8]) # Extends the list with [7, 8]

d. Removing Elements

Example:

my_list.remove(3) # Removes the first occurrence of 3

popped_element = my_list.pop() # Removes and returns the last element

2. String

- A string is a sequence of characters, enclosed in either single (`'`) or double (`"`) quotes.
- Strings are immutable, which means once created, they cannot be changed.
- You can perform various string operations like concatenation, slicing, and formatting.
- Example:

  my_string = "Hello, World!"

  name = "Alice"

  greeting = f"Hello, {name}!"

Some Basic String operations

a. String Concatenation

Example:

greeting = "Hello, " + "Alice!"

b. String Length

Example:

length = len(my_string) # Returns the length of the string

c. Substring

Example:

substring = my_string[7:12] # Extracts "World"

d. String Methods

Example:

uppercase_string = my_string.upper() # Converts to uppercase

3. Set

- A set is an unordered collection of unique elements.
- Sets are defined using curly braces `{}` or the `set()` constructor.
- Sets are commonly used for tasks that involve testing membership or eliminating duplicates.
- Example:

  my_set = {1, 2, 3, 4, 5}

  unique_characters = set("hello")

Some Basic Set operations

a. Adding and Removing Elements

Example:

my_set.add(6) # Adds 6 to the set

my_set.remove(3) # Removes 3 from the set

b. Set Operations:

Example:

set1 = {1, 2, 3}

set2 = {3, 4, 5}

union_set = set1.union(set2) # Union of sets (1, 2, 3, 4, 5)

intersection_set = set1.intersection(set2) # Intersection (3)

4. Dictionary

- A dictionary is an unordered collection of key-value pairs.
- Each key in a dictionary is unique and maps to a specific value.
- Dictionaries are defined using curly braces `{ }` with key-value pairs separated by colons ` : `.
- Example:

  my_dict = {"name": "John", "age": 30, "city": "New York"}

  student_scores = {"Alice": 95, "Bob": 87, "Charlie": 92}

Some Basic Set operations

a. Accessing Values:

<u>Example:</u>

name = my_dict['name'] # Access the value associated with 'name'

b. Adding and Updating Key-Value Pairs:

<u>Example:</u>

my_dict['occupation'] = 'Engineer' # Add a new key-value pair

my_dict['age'] = 31 # Update the value associated with 'age'

c. Removing Key-Value Pairs:

Example:

del my_dict['city'] # Removes the 'city' key and its value

These data types are fundamental in Python and are used extensively in various programming tasks. Understanding when and how to use them is crucial for effective Python programming.

5. **Bubble Sort Algorithm:**

Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process continues until the list is sorted. It's called "Bubble Sort" because the smaller (or larger, depending on the sorting order) elements "bubble" to the top of the list with each pass.

Here's a description of Bubble Sort in Python along with an example:

<u>Bubble Sort Algorithm Steps:</u>

a. Start at the beginning of the list.

b. Compare the first two elements. If the first element is greater (or smaller, depending on the sorting order) than the second, swap them.

c. Move one position to the right.

d. Repeat steps 2-3 until you reach the end of the list.

e. Continue this process for each pair of adjacent elements, moving from the beginning to the end of the list.

f. Repeat steps 1-5 until no more swaps are needed, indicating that the list is sorted.

1. **To merge two lists and find the second largest element in the list using bubble sort.**

   Code:

   ```
   def bubble_sort(arr):
     n = len(arr)
     for i in range(n):
       for j in range(0, n - i - 1):
         if arr[j] > arr[j + 1]:
           arr[j], arr[j + 1] = arr[j + 1], arr[j]


   list1 = [1,3,8,9]
   list2 = [7,4,7,6]


   merged_list = list1 + list2
   bubble_sort(merged_list)
   second_largest = merged_list[-2]
   print("Merged List:", merged_list)
   print("Second Largest Element:", second_largest)
   ```

   Conclusion:

   The code successfully merges two lists (list1 and list2) into a single list (merged_list) and sorts it using the Bubble Sort algorithm. After sorting, it finds and prints the second largest element in the sorted merged list. Please note that Bubble Sort is not the most efficient sorting algorithm, especially for large lists, but it serves as an example for educational purposes.

   Output:

   ```
   PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS

   [Running] python -u "f:\Pushkar\MCA\Sem-1\Python Programming\largest_element_bubble.py"
   Merged List: [1, 3, 4, 6, 7, 7, 8, 9]
   Second Largest Element: 8

   [Done] exited with code=0 in 0.062 seconds
   ```

2. **To calculate the no of uppercase, lowercase letters and digits in a string.**

   Code:

```
input_string = "Hello World 12345"
upper_count=0
lower_count=0
digit_count=0
for char in input_string:
  if char.isupper():
    upper_count+=1
for char in input_string:
  if char.islower():
    lower_count+=1
for char in input_string:
  if char.isdigit():
    digit_count+=1
print("Uppercase letters:", upper_count)
print("Lowercase letters:", lower_count)
print("Digits:", digit_count)
```

   Conclusion:

   We have learned how to analyze a given input string in Python and count the occurrences of uppercase letters, lowercase letters, and digits using string methods and loops.

   Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS

[Running] python -u "f:\Pushkar\MCA\Sem-1\Python Programming\calculate_letter.py"
Uppercase letters: 2
Lowercase letters: 8
Digits: 5

[Done] exited with code=0 in 0.058 seconds
```
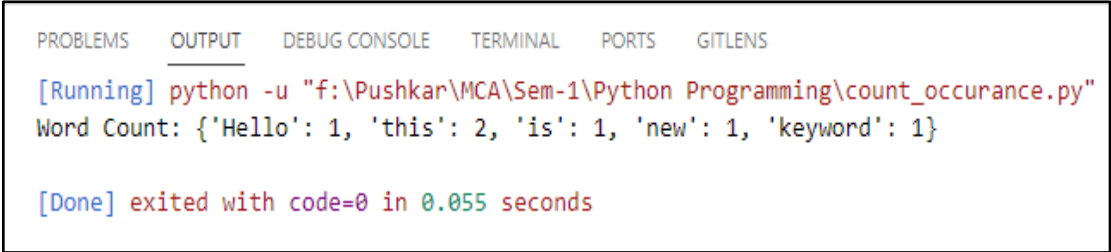
3. **To count the occurrences of each word in a given string sentence.**

   Code:

   sentence = "Hello this is new my keyword"

   words = sentence.split()

   word_count = {}

   for word in words:

       word_count[word] = word_count.get(word, 0) + 1

   print("Word Count:", word_count)

   Conclusion:

   In this code, we've learned how to tokenize a sentence into words, create a dictionary to count the frequency of each word, and print the word frequency count. This is a fundamental operation often used in natural language processing and text analysis tasks.

   Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS

[Running] python -u "f:\Pushkar\MCA\Sem-1\Python Programming\count_occurance.py"
Word Count: {'Hello': 1, 'this': 2, 'is': 1, 'new': 1, 'keyword': 1}

[Done] exited with code=0 in 0.055 seconds
```

4. **To add a key value pair to the dictionary and search and then delete the given key from the dictionary.**

Code:

```
my_dict = {}
my_dict['name'] = 'ABC'
my_dict['age'] = 31
my_dict['city'] = 'New York'
print("Dictionary after adding key-value pairs:")
print(my_dict)
search_key = 'age'
if search_key in my_dict:
  print(f"The value for key '{search_key}' is: {my_dict[search_key]}")
else:
  print(f"Key '{search_key}' not found in the dictionary")
delete_key = 'city'
if delete_key in my_dict:
  del my_dict[delete_key]
  print(f"Key '{delete_key}' deleted from the dictionary")
else:
   print(f"Key '{delete_key}' not found in the dictionary")
print("Dictionary after deleting a key:")
print(my_dict)
```

Conclusion:

In this code, we've learned essential dictionary operations, including adding, searching for, and deleting key-value pairs. This demonstrates how dictionaries are useful for organizing and manipulating data in Python.

Output:


```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS

[Running] python -u "f:\Pushkar\MCA\Sem-1\Python Programming\dict_op.py"
Dictionary after adding key-value pairs:
{'name': 'ABC', 'age': 31, 'city': 'New York'}
The value for key 'age' is: 31
Key 'city' deleted from the dictionary
Dictionary after deleting a key:
{'name': 'ABC', 'age': 31}

[Done] exited with code=0 in 0.059 seconds
```

5. **Create one dictionary of 5 students with their name, address, age, class and marks of 5 subjects. Perform all the operations on the created dictionary.**

Code:

```
# Create a dictionary for 5 students
students = {
    'student1': {
        'name': 'ABC',
        'address': 'MU',
        'age': 18,
        'class': '11th',
        'marks': {
            'Mathematics': 85,
            'science': 92,
            'history': 78,
            'english': 88,
            'Drawing': 95
        }
    },

    'student2': {
        'name': 'DEF',
        'address': 'NA',
        'age': 17,
```

```
        'class': '6th',
        'marks': {
            'Mathematics': 90,
            'science': 88,
            'history': 76,
            'english': 91,
            'Drawing': 84
        }
    },

    'student3': {
        'name': 'GHI',
        'address': 'CH',
        'age': 19,
        'class': '10th',
        'marks': {
            'Mathematics': 78,
            'science': 85,
            'history': 92,
            'english': 80,
            'Drawing': 89
        }
    },

    'student4': {
        'name': 'JKL',
        'address': 'MP',
        'age': 17,
        'class': '8th',
        'marks': {
            'Mathematics': 92,
            'science': 84,
            'history': 76,
            'english': 90,
```

```
            'Drawing': 82
        }
    },

    'student5': {
        'name': 'MNO',
        'address': 'AG',
        'age': 18,
        'class': '9th',
        'marks': {
            'Mathematics': 88,
            'science': 90,
            'history': 85,
            'english': 87,
            'Drawing': 91
        }
    }
}

print("Details of All the Students :")
# Display the information for each student
for student_id, student_info in students.items():
    print(f"Student ID: {student_id}")
    print(f"Name: {student_info['name']}")
    print(f"Address: {student_info['address']}")
    print(f"Age: {student_info['age']}")
    print(f"Class: {student_info['class']}")
    print("Marks:")

for subject, marks in student_info['marks'].items():
    print(f"{subject}: {marks}")
    print()
```

```
# Search for a student by ID
search_id = 'student3'
if search_id in students:
    print(f"Student ID: {search_id}")
    student_info = students[search_id]
    print(f"Name: {student_info['name']}")
else:
    print(f"Student with ID '{search_id}' not found")


# Delete a student by ID
delete_id = 'student4'
if delete_id in students:
    del students[delete_id]
    print(f"Student with ID '{delete_id}' deleted from the dictionary")
else:
    print(f"Student with ID '{delete_id}' not found in the dictionary")
```

Conclusion:

In this code, we've learned about dictionaries, nested dictionaries, and how to manipulate and access data within them. It's a practical example of organizing and managing structured data in Python, which is essential for various data processing and management tasks.

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS

[Running] python -u "f:\Pushkar\MCA\Sem-1\Python Programming\stud_dict.py"
Details of All the Students :
Student ID: student1
Name: ABC
Address: MU
Age: 18
Class: 11th
Marks:
Student ID: student2
Name: DEF
Address: NA
Age: 17
Class: 6th
Marks:
Student ID: student3
Name: GHI
Address: CH
Age: 19
Class: 10th
Marks:
Student ID: student4
Name: JKL
Address: MP
Age: 17
Class: 8th
Marks:
Student ID: student5
Name: MNO
Address: AG
Age: 18
Class: 9th
Marks:
Mathematics: 88

science: 90

history: 85

english: 87

Drawing: 91

Student ID: student3
Name: GHI
Student with ID 'student4' deleted from the dictionary

[Done] exited with code=0 in 0.073 seconds
```
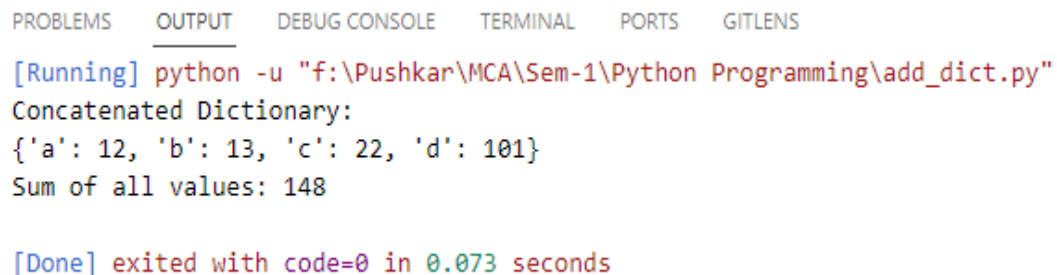
6. **To concatenate two dictionaries and find the sum of all values in the dictionary.**

    Code:

    # Define two dictionaries

    dict1 = {'a': 12, 'b': 59, 'c': 71}

    dict2 = {'b': 13, 'c': 22, 'd': 101}

    # Concatenate the two dictionaries

    concatenated_dict = {**dict1, **dict2}

    # Calculate the sum of all values in the concatenated dictionary

    total_sum = sum(concatenated_dict.values())

    # Display the concatenated dictionary and the sum

    print("Concatenated Dictionary:")

    print(concatenated_dict)

    print(f"Sum of all values: {total_sum}")

    Conclusion:

    In this code, we've learned how to merge two dictionaries into a single concatenated dictionary and calculate the sum of values within it. This is a useful technique for combining data from multiple sources and performing aggregate operations on the merged data

    Output:

    ```
    PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS

    [Running] python -u "f:\Pushkar\MCA\Sem-1\Python Programming\add_dict.py"
    Concatenated Dictionary:
    {'a': 12, 'b': 13, 'c': 22, 'd': 101}
    Sum of all values: 148

    [Done] exited with code=0 in 0.073 seconds
    ```

7. To add and remove elements from set and perform all the set operations like Union, Intersection, Difference and Symmetric Difference.

Code:

```
# Create two sets
set1 = {2, 3, 4, 5, 6}
set2 = {4, 5, 6, 7, 8}
# Add an element to a set
set1.add(6)
print("After adding 6 to set1:", set1)
# Remove an element from a set
set2.remove(7)
print("After removing 7 from set2:", set2)
# Union of two sets
union_result = set1.union(set2)
print("Union of set1 and set2:", union_result)
# Intersection of two sets
intersection_result = set1.intersection(set2)
print("Intersection of set1 and set2:", intersection_result)
# Difference of two sets
difference_result = set1.difference(set2)
print("Difference of set1 and set2:", difference_result)
# Symmetric Difference of two sets
symmetric_difference_result = set1.symmetric_difference(set2)
print("Symmetric Difference of set1 and set2:", symmetric_difference_result)
```

Conclusion:

In this code, we've learned how to perform common set operations such as adding, removing, finding the union, intersection, difference, and symmetric difference of sets. Sets are useful for dealing with unique and unordered collections of elements in Python.

Output:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   GITLENS

[Running] python -u "f:\Pushkar\MCA\Sem-1\Python Programming\set_op.py"
After adding 6 to set1: {2, 3, 4, 5, 6}
After removing 7 from set2: {4, 5, 6, 8}
Union of set1 and set2: {2, 3, 4, 5, 6, 8}
Intersection of set1 and set2: {4, 5, 6}
Difference of set1 and set2: {2, 3}
Symmetric Difference of set1 and set2: {2, 3, 8}

[Done] exited with code=0 in 0.058 seconds
```

8. **Perform different operations on Tuple.**

   Code:

   # Creating a tuple

   my_tuple = (10, 11, 12, 13, 14)

   # Accessing elements

   print("Accessing elements:")

   print(my_tuple[0]) # Access the first element (1)

   print(my_tuple[-1]) # Access the last element (5)

   # Slicing

   print("\nSlicing:")

   sliced_tuple = my_tuple[1:4] # Creates a new tuple (2, 3, 4)

   print(sliced_tuple)

   # Concatenating tuples

   print("\nConcatenating tuples:")

   tuple1 = (1, 2)

   tuple2 = (3, 4)

   concatenated_tuple = tuple1 + tuple2 # Creates a new tuple (1, 2, 3, 4)

   print(concatenated_tuple)

   # Tuple repetition

   print("\nTuple repetition:")

   repeated_tuple = my_tuple * 2 # Creates a new tuple (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)

   print(repeated_tuple)

```
# Finding tuple length
print("\nFinding tuple length:")
length = len(my_tuple) # Returns 5
print(length)
# Iterating through a tuple
print("\nIterating through a tuple:")
for item in my_tuple:
    print(item)
# Checking membership
print("\nChecking membership:")
if 12 in my_tuple:
    print("3 is in the tuple")
# Tuple unpacking
print("\nTuple unpacking:")
a, b, c, d, e = my_tuple
print(f"a: {a}, b: {b}, c: {c}, d: {d}, e: {e}")
# Count and index
print("\nCount and index:")
count = my_tuple.count(3) # Returns the count of 3 in the tuple
index = my_tuple.index(12) # Returns the index of the first occurrence of 4
print(f"Count of 3: {count}")
print(f"Index of 4: {index}")
```

Conclusion:

In this code, we've learned various operations and techniques for working with tuples in Python. Tuples are immutable, ordered collections of elements and can be useful in scenarios where data should not be modified after creation.

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS

[Running] python -u "f:\Pushkar\MCA\Sem-1\Python Programming\tuple_op.py"
Accessing elements:
10
14

Slicing:
(11, 12, 13)

Concatenating tuples:
(1, 2, 3, 4)

Tuple repetition:
(10, 11, 12, 13, 14, 10, 11, 12, 13, 14)

Finding tuple length:
5

Iterating through a tuple:
10
11
12
13
14

Checking membership:
3 is in the tuple

Tuple unpacking:
a: 10, b: 11, c: 12, d: 13, e: 14

Count and index:
Count of 3: 0
Index of 4: 2

[Done] exited with code=0 in 0.063 seconds
```

9. **Write a Python program to count the elements in a list until an element is a tuple.**

   Code:

   my_list = [11, 22, 33, 'world', (43, 54), 62, 71]

   # Initialize a counter
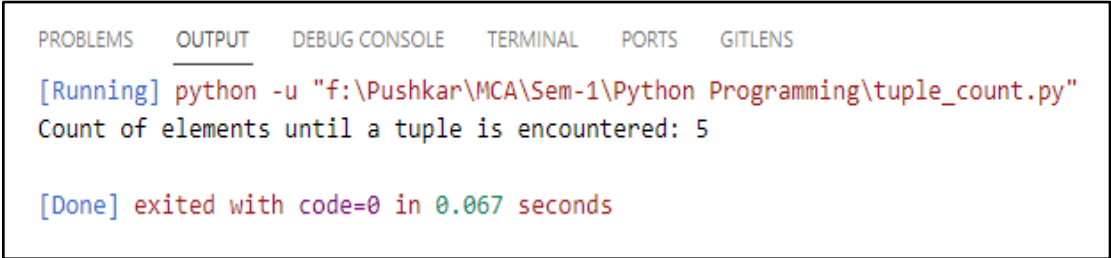
   count = 0

   # Iterate through the list

   for item in my_list:

```
   count += 1
  if isinstance(item, tuple):
   break
```

```python
# Print the count of elements until a tuple is encountered
print(f"Count of elements until a tuple is encountered: {count}")
```

Conclusion:

In this code, we've learned how to iterate through a list and count the number of elements until a specific condition is met (in this case, until a tuple is encountered). This demonstrates how to use a for loop, conditionals, and the break statement for control flow in Python.

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS

[Running] python -u "f:\Pushkar\MCA\Sem-1\Python Programming\tuple_count.py"
Count of elements until a tuple is encountered: 5

[Done] exited with code=0 in 0.067 seconds
```